SVEUČILIŠTE U ZAGREBU
FILOZOFSKI FAKULTET
ODSJEK ZA INFORMACIJSKE I KOMUNIKACIJSKE ZNANOSTI

Ak. god. 2014./2015.

Petar-Krešimir Marić

NoSQL baze podataka

diplomski rad

Mentor: doc. dr. sc. Krešimir Pavlina

Zagreb, 2015.

Sadržaj

1. Uvod....3

2. Što je NoSQL?....4
    2.1. NoSQL pokret....6

3. Vrste NoSQL baza....7
    3.1. Key-value baze....7
    3.2. Column-family baze....8
    3.3. Document baze....9
    3.4. Graph baze....9

4. Modeli podataka....11
    4.1. Usporedba ACID i BASE modela....12
        4.1.1. ACID....13
        4.1.2. BASE....14

5. Modeli distribucije....17
    5.1. Konzistentnost....19

6. Pregled NoSQL baza....22
    6.1. Riak....22
    6.2. MongoDB....24
    6.3. Cassandra....27
    6.4. Neo4j....30

7. Zaključak....35

8. Literatura....36

1. Uvod

Informacijska tehnologija se brzo mijenja, pogotovo posljednjih godina. Izmijenjuju se programski jezici, arhitektura, platforme i procesi, međutim jedna stvar ostaje nepromijenjena – relacijske baze podataka su se uvijek bile prisutne u manje-više istom obliku već više od trideset godina. Većina organizacija se oslanja na baze podataka za svoje djelovanje i poslovanje, ne samo za unutarnje operacije nego i za interakciju preko mreže sa svojim kupcima, klijentima i dobavljačima. Sustavi za upravaljanje bazama podataka su dosta kompleksni i osiguravaju visok stupanj učinkovitosti, dostupnosti i sigurnosti podataka, posebno kada pristup bazi ima veliki broj korisnika.

Ubrzani razvoj interneta, pogotovo pojava weba 2.0 i društvenih mreža, pružio je izazov relacijskim bazama podataka. Stroga struktura relacijskog modela nije najbolje odgovarala ogromnoj količini nestrukturiranih podataka na webu, koje je bilo teško pohraniti u bazu na učinkovit način, osiguravajući dostupnost i sigurnost.
Posljednjih godina se pojavila nova vrsta baze podataka nazvana NoSQL (engl. Not Only SQL) koja je nastala je zbog potrebe za pohranom rastuće količine podataka što je promijenilo način na koji se gleda sustave za pohranu.

Naziv „NoSQL“ nije strogo definiran. Upotrijebljava se kod opisivanja raznih baza podataka koje nisu bazirane na relacijskom modelu. Obuhvaćaju podatke bez sheme (engl. schemaless data), vrte se na klasterima (engl. cluster) te umjesto tradicionalne konzistentnosti stavljaju prioritet na druga svojstva. Zagovaratelji NoSQL-a često govore da se NoSQL-om mogu raditi sustavi koji imaju bolje preformanse, veću skalabilnost i lakši su za programiranje.

2. Što je NoSQL?

Postoje dobri razlozi zbog kojeg su relacijske baze toliko dugo standard u pohrani podataka. Sadrže karakteristike koje osiguravaju postojanost podataka, odnosno sigurnu pohranu i pristup podacima, transakcije, koje osiguravaju da istodoban pristup podacima s više strana ne uzrokuje probleme, mogu se integrirati s više aplikacija tako da aplikacije spremaju podatke u istu bazu, te postoji standardni model; iako neke razlike između pojedinih baza postoje, osnovni mehanizmi su isti i mogu se primjeniti na sve relacijske baze.

2000-ih godina promet na webu počeo se ubrzano povećavati. Također, postalo je moguće pratiti razne vrste podataka, pogotovo aktivnosti korisnika. Snalaženje s rastom podataka i korisnika zahtijevalo je više resursa, što je značilo izbor između dva načina skaliranja: vertikalnog (skaliranje prema gore) i horizontalnog (skaliranje prema van). Vertikalno skaliranje zahtijeva veće strojeve, brže procesore, više memorije i više mjesta za pohranu. To je značilo i puno veću cijenu, ali su postojala i fizička ograničenja. S druge strane, horizontalno skaliranje se dobiva postavljanjem više manjih strojeva u klaster, što je financijski manje zahtjevno, ali potencijalno znači i veću izdržljivost sustava jer dok poneki stroj može biti u kvaru, klaster će i dalje raditi što pruža visoku pouzdanost.

Međutim, relacijske baze nisu namijenjene radu na klasterima. Moguće je koristiti *sharding*, gdje su podaci „razlomljeni“ i raspodijeljeni po različitim serverima, što smanjuje opterećenje, ali se time između shardova ("krhotina" podataka) gube gotovo sve prednosti relacijskog modela: upiti, referencijalni integritet, transakcije i konzistentnost.[1]

Problemi relacijskih baza u radu s klasterima ponukali su neke organizacije da potraže alternativna rješenja. Google i Amazon bili su jako utjecajni na tom polju. Njihove baze, nazvane BigTable i Dynamo pokrenule su novi val u metodama pohrane podataka.

Izraz NoSQL se proširio i počeo koristiti za baze poput BigTable-a i Dynamo-a, ali usprkos tome nema striktnu definiciju. Samo definiranje termina NoSQL je izazovno s obzirom da se radi o prilično širokom pojmu. McCreary i Kelly (2014.) definiraju NoSQL kao "skup koncepta koji omogućavaju brzu i učinkovitu obradu podataka s fokusom na preformanse,

[1] Attack of the Clusters (1.4.). Sadalage, P.J.; Fowler, M. NoSQL Distilled. Addison-Wesley. 2013.

pouzdanost i agilnost."[2] Ta definicija sama po sebi ne isključuje relacijske baze, zbog čega je bitno razumjeti tematiku NoSQL pokreta i što on predstavlja.

NoSQL sustavi uglavnom imaju određene zajedničke karakteristike koje ih povezuju. Najznačajnija je nedostatak SQL-a kao jezika za upite. Neke baze imaju zamjenski jezik, poput Cassandrinog CQL-a koji je relativno sličan. NoSQL baze su uglavnom open-source projekti, iako postoje iznimke. Relacijske baze koriste ACID transakcije za održavanje konzistentnosti što nije pogodno za rad na klasterima, stoga NoSQL baze nude raspon opcija za razinu konzistentnosti i način distribucije podataka. Iako je većina NoSQL sustava rađena za potrebe klastera, postoje iznimke poput graf baza koje koriste distribucijski model sličan relacijskim bazama, ali s drugačijim modelom podataka koji je prilagođen kompleksnim poveznicama između podataka. NoSQL baze nemaju shemu, što znači da dopuštaju dodavanje novih polja bez prethodnog definiranja strukture. To je korisno kod spremanja nestrukturiranih i promjenjivih vrsta podataka. NoSQL baze rađene su za potrebe web sjedišta 21. stoljeća, tako da se samo sustavi razvijani u tom vremenskom periodu nazivaju NoSQL-om.

Navedene karakteristike povezuju NoSQL baze pod istim nazivom, ali to nije striktna definicija i moguće da će još dolaziti do promjena i proširenja značenja NoSQL-a. Stoga je najbolje promatrati NoSQL kao pokret umjesto kao naziv za određenu tehnologiju.

Postoje četiri osnovne vrste baza koje se svrstavaju pod NoSQL kategoriju:

- **key-value** - jednostavna baza koja pohranjuje podatke pomoću parova ključeva i njihovih vrijednosti
- **column-family** - baza koja koristi tablice odnosno pohranjuje vrijednosti u redove i stupce
- **document** - pohranjuje strukture podataka koje imaju hijerarhiju
- **graph** - sprema jednostavne podatke s kompleksnim međusobnim poveznicama prikazanih grafom

Standardizirani SQL sustavi omogućavaju da se aplikacije prenose između baza koje koriste zajednički jezik. NoSQL sustavi još nisu standardizirani, ali bi pritisak tržišta mogao

[2] What is NoSQL (1.1.). McCreary, D; Kelly, A. Making Sense of NoSQL. Manning. 2014.

primorati razvoj standarda koji bi omogućio sličan stupanj prenosivosti kao kod relacijskih baza.[3]

2.1. NoSQL pokret

McCreary i Kelly (2014.) identificiraju 4 osnovna svojstva koja su pokrenula NoSQL: Volumen, brzina, promjenjivost i agilnost.[4] Volumen se odnosi na količinu podataka koji zahtjevaju obradu i pohranu u bazu, odnosno povećao se toliko da vertikalna skalabilnost više nije bila moguća, stoga su bili potrebni sustavi koji omogućuju horizontalnu skalabilnost i paralelnu obradu podataka. Brzina se odnosi na potrebu da s povećanjem količine podataka i online zahtjevima, podatke je potrebno i obraditi zadovoljavajućom brzinom kako ne bi došlo do usporavanja sustava. Promjenjivost ukazuje na činjenicu da krute sheme relacijskih baza nisu pogodne čestim promjenama, koje su skupe i zahtjevaju vremena. Agilnost znači da je potreban jednostavan unos podataka u bazu i dohvat podataka iz baze, što kod relacijskih sustava može biti kompleksno ovisno o strukturi podataka.

Ukratko, pojavila se potreba za novim načinom obrade podataka. NoSQL je nova paradigma koja zahtjeva drugačiji način razmišljanja i rješavanja problema. NoSQL tehnologije su prilagodljivije promjenama i određenim zahtjevima tržišta.

[3] NoSQL business drivers (1.2.). McCreary, D; Kelly, A. Making Sense of NoSQL. Manning. 2014.

[4] NoSQL business drivers (1.2.). McCreary, D; Kelly, A. Making Sense of NoSQL. Manning. 2014.

3. Vrste NoSQL baza

Iako NoSQL baze mogu jako međusobno varirati, moguće je razaznati četiri osnovna tipa: key-value (engl. ključ-vrijednost), column-family (engl. obitelj stupaca), document (engl. dokumentne) i graph (engl. graf).

3.1. Key-value baze

Key-value baza je jednostavna baza koja pomoću stringa (koji je ključ) vraća BLOB podataka (engl. Binary Large OBject – skupina binarnih podataka spremljenih u bazu kao jedan entitet). Nema jezik za upite, nego je moguće samo dodavanje parova ključeva i vrijednosti u bazu, dohvaćanje neke vrijednosti na temelju ključa, te brisanje podataka iz baze. McCreary i Kelly uspoređuju key-value bazu s rječnikom, odnosno rječnik se može smatrati key-value bazom u kojoj riječi predstavljaju ključeve dok su definicije njihove vrijednosti.[5] Riječi su poredane po abecedi, pa je pronalazak traženog pojma vrlo brz, tj. nije potrebno pregledati cijeli rječnik kako bi se došlo do tražene definicije. Key-value baze funkcioniraju na istom principu. Indeksiraju se po ključu, stoga se vrlo brzo dolazi do traženog podatka, bez obzira na veličinu baze.

Nisu specificirani tipovi podataka koji se mogu pohraniti. Prednost toga je što se u bazu može pohraniti bilo koji oblik podataka kao vrijednost. Sustav će pohraniti vrijednost kao BLOB i vratiti ga kada dobije zahtjev za čitanje. O aplikaciji ovisi koji tip podataka se pohranjuje u bazu. Format za ključeve je također fleksibilan i može se sastojati od gotovo bilo kakvog stringa.

Key-value baza može se zamisliti kao tablica od dva stupca u kojoj prvi stupac predstavlja ključ, a drugi vrijednost. Postoje tri operacije koje se mogu izvršiti nad bazom: PUT, GET i DELETE. PUT postavlja novi par ključ-vrijednost ili ažurira vrijednost ako ključ već postoji. GET vraća vrijednost na temelju zadanog ključa. DELETE briše ključ i njegovu vrijednost iz baze.

[5] What is a key-value store? (4.1.1.). McCreary, D; Kelly, A. Making Sense of NoSQL. Manning. 2014.

Postoje dva osnovna svojstva key-value baza: Jedinstveni ključevi i nemogućnost upita. Svi ključevi moraju biti jedinstveni, odnosno nije moguće imati dva ista ključa. Nije moguće izvršavati upite nad vrijednostima u bazi, jer su operacije moguće samo nad ključevima. Nema previše ograničenja u vezi toga što se može iskoristiti kao ključ, dok god je string razumne duljine. Isto tako, spremanje vrijednosti je ograničeno jedino veličinom prostora za pohranu. Dok god u bazi ima dovoljno mjesta može se pohraniti bilo što, uključujući i razne multimedijske sadržaje.

3.2. Column-family baze

Column-family baze koriste identifikatore redova i stupaca kao ključeve za pronalazak podataka. Mogu se ugrubo usporediti sa proračunskim tablicama u kojima se koristi kombinacija broja reda i slova stupca kako bi se pronašla određena ćelija koja sadrži vrijednost. Kao i kod key-value baza, ćelija u column-family bazi može sadržavati gotovo bilo koju vrijednost. Column-family baze se mogu lako skalirati kako bi pohranili veće količine podataka.

U velikim implementacijama moguće je imati bazu od stotinjak tisuća redova i stupaca, ali neće svaka ćelija biti ispunjena, dapače, moguće je da samo mali postotak ćelija sadrži neku vrijednost. Relacijske baze se teško nose s ovako "prozračnim" tablicama, ali column-family baze su dizajnirane upravo s ovom namjerom. Kako bi se lakše pregledali stupci koji sadrže povezane podatke, oni se grupiraju u obitelj stupaca, iz čega dolazi naziv za bazu.

Kao i kod key-value baza, nije potrebno unaprijed definirati model podataka prije nego što se podaci unesu u bazu, ID redova i nazivi stupaca se mogu odrediti bilo kada, no trebalo bi odlučiti kako grupirati podatke u obitelj stupaca kako bi se mogao napraviti ključ za tu obitelj stupaca.

Column-family baze su dizajnirane za rad na distribuiranim klasterima i nisu prigodne za male setova podataka. Obično je potrebno minimalno 5 procesora kako bi se opravdao klaster, s obzirom na to da su mnoge Column-family baze dizajnirane su tako da spremaju podatke na barem tri čvora za potrebe replikacije.[6]

---

[6] Benefits of column family systems (4.3.3.). McCreary, D; Kelly, A. Making Sense of NoSQL. Manning. 2014.

3.3. Document baze

Dokumente baze su najfleksibilnije i najpopularnije NoSQL baze.[7] Kao i key-value baze, dokumentne baze koriste ključ koji je vezan za određenu vrijednost. No dok su vrijednosti u key-value bazi zatvorene odnosno nevidljive - ne možemo im nikako pristupiti osim pomoću ključa - vrijednosti u dokumentnoj bazi podataka se mogu indeksirati i pretraživati. Svi podaci unutar dokumenta se indeksiraju pri njegovom dodavanju u bazu. Ako znamo jedno svojstvo nekog dokumenta, možemo pronaći i ostale dokumente s istim svojstvom. Također, moguće je pronaći točnu lokaciju traženog podatka unutar dokumenta koristeći „document path" – vrstu ključa koja otvara pristup stablastoj strukturi dokumenta. Moguće je i vratiti samo određeni dio dokumenta kojeg tražimo bez učitavanja cijelog dokumenta, npr. možemo pronaći paragraf iz knjige bez učitavanje cijele knjige.

Dokumenti se grupiraju u kolekcije odnosno zbirke kako bi se lakše upravljalo rastućom bazom. Kolekcije su usporedive sa strukturom direktorija u Windowsima. Mogu poslužiti za lakšu navigaciju hijerarhijskom strukturom dokumenata, grupiranje dokumenata po sličnosti ili određenom kriteriju i sl. Kolekcije mogu unutar sebe sadržavati druge kolekcije.

3.4. Graph baze

Graf baze su sustavi koji sadrže slijed čvorova i poveznica koji čine graf. Optimizirane su za pohranu čvorova i njihovih međusobnih poveznica, te izvršavanje upita nad njima.[8] Čvorovi u graf bazi obično predstavljaju objekte, koji mogu biti ljudi, organizacije, web stranice, računala u mreži i sl. Poveznice su veze između tih objekata i obično se predstavljaju linijama koje spajaju dva čvora. Upiti za graf baze se obično predočavaju kao putovanje po grafu (engl. graph traversal) kako bi došli do tražene informacije (koja može biti npr. pronalazak najkraćeg puta između dva čvora kako bi otkrili podatak koji ih povezuje).

Ovaj tip baza je idealan ako imamo mnogo podataka koji imaju kompleksnu strukturu međusobnih veza. Možemo jednostavnim upitima pronaći najbliže susjedne čvorove, ali i

[7] Document stores (4.4.). McCreary, D; Kelly, A. Making Sense of NoSQL. Manning. 2014.

[8] Graph stores (4.2.). McCreary, D; Kelly, A. Making Sense of NoSQL. Manning. 2014.

pronaći uzorke u mreži poveznica. Poveznice među čvorovima mogu sadržavati razna svojsta, pa nam tako baza može pružiti detaljne izvještaje o njihovoj povezanosti.

Graf baze su iznimno korisne u aplikacijama koje imaju potrebu analizirati veze između objekata ili proći kroz sve čvorove određenim redosljedom. Koriste se kad je potrebno pohraniti kompleksne veze među objektima, npr. u društvenim mrežama ili sustavima koji trebaju brzo analizirati kompleksnu strukturu mreže i pronaći uzorke u njoj. Za razliku od većine NoSQL baza, graf baze podržavaju ACID model.

4. Modeli podataka

Jedna od ključnih karakteristika NoSQL baza je odbacivanje relacijskog modela. Svaka NoSQL baza koristi drugačiji model podataka, no mogu se grupirati u četiri glavne skupine: key-value, document, column-family i graph baze. Sadalage i Fowler prve tri vrste baza nazivaju agregatno orijentiranima (engl. aggregate-oriented) zbog sličnih svojstava.[9] Relacijski model dijeli informaciju koju pohranjujemo u n-torke. One imaju ograničenu strukturu i ta jednostavnost je ključna za relacijski model. Agregatno orijentirane baze imaju drugačiji pristup jer je potrebno obraditi podatke sa kompleksnijom strukturom. Agregat je u ovom slučaju samo skupina povezanih objekata koja se tretira kao jedna jedinica. Key-value, document i column-family baze koriste takav model. Agregati olakšavaju rad baze na klasterima i prigodni su za replikaciju i sharding. Bazi možemo dati do znaja koji podaci trebaju biti pohranjeni zajedno na istom klasteru kako bi se lakše manipuliralo njima. Relacijski model, odnosno ACID omogućuje manipuliranje više redova iz raznih tablica unutar jedne transakcije, kao jedna operacija. Agregatno orijentirane baze nemaju ACID svojstva pa samim time ne mogu izvoditi operaciju nad više agregata odjednom. Ukoliko želimo izvršiti operaciju nad više agregata odjednom, moramo to napraviti preko aplikacije. Zbog toga je bitno razmisliti o tome kako se i koji podaci grupiraju zajedno u jedan agregat.

Za razliku od agregatno orijentiranih baza, koje su rađene za kompleksne zapise s relativno jednostavnim poveznicama, graf baze su fokusirane na suprotni problem: jednostavnije podatke sa kompleksnim međusobnim vezama. To može biti društvena mreža, u kojoj čvorovi sadrže malo podataka (npr. ime osobe), ali ti čvorovi imaju bogatu strukturu poveznica. Ovaj model podataka razlikuje se od agregatno orijentiranih baza. Ovakve baze najbolje rade na jednom serveru za razliku od agregatnih baza koje su rađene za distribuciju na klasterima. Također, graf baze podržavaju ACID transakcije koje su potrebne za izvršavanje upita nad više čvorova i poveznica odjenom, bez gubitka konzistentnosti.[10] Zapravo je jedina sličnost graf baza i ostalih NoSQL baza u odbacivanju SQL-a i vremenu pojavljivanja, stoga ih se obično stavlja u istu skupinu.

[9] Aggregates (2.1.). Sadalage, P.J.; Fowler, M. NoSQL Distilled. Addison-Wesley. 2013.

[10] Graph Databases (3.2.). Sadalage, P.J.; Fowler, M. NoSQL Distilled. Addison-Wesley. 2013.

Još jedna značajna karakteristika NoSQL baza je nedostatak sheme. Relacijska baza mora imati definiranu strukturu, tablice, stupce, te vrstu podataka koju svaki stupac može sadržavati. Pohranjivanje podataka u NoSQL baze nije tako strogo definirano. Key-value baze i document baze mogu pohraniti bilo kakav oblik podatka kojemu se onda dodijeljuje ključ. Column-family baze dozvoljavaju spremanje bilo kakvog podatka pod bilo kojim stupcem. Graf bazama se mogu bez problema dodavati novi čvorovi sa svojim svojstvima. Ovakva sloboda omogućava lako izmjenjivanje baze kada je potrebno. Isto tako, ukoliko neki podaci više nisu potrebni, mogu se slobodno obrisati, bez brige oko gubitka drugih podataka kao što bi bilo s brisanjem stupaca u relacijskoj bazi. Nedostatak sheme također olakšava pohranu nejednolikih podataka (u kojem svaki zapis ima različita polja).

Usprkos tome što NoSQL baze ne sadržavaju shemu, ona ipak postoji, ali nalazi se u aplikacijskom sloju odnosno kodu aplikacije koja pristupa bazi. Ovakav pristup oslobađa rasterećuje bazu, ali može biti problematičan ako više različitih aplikacija pokušava pristupiti podacima, pogotovo ako su aplikacije od različitih developera.

Iako se nedostatak sheme i fleksibilnost koju to donosi smatra pozitivnim aspektom kod NoSQL baza, Sadalage i Fowler ukazuju na činjenicu da relacijske sheme nisu toliko nefleksibilne kao što zagovornici NoSQL-a tvrde.[11] Relacijska shema može se promjeniti standardnim SQL naredbama. No to je isplativo jedino ukoliko nema prečestih promjena strukture podataka ili potrebe za spremanjem velike količine nejednolikih podataka, za što je nedostatak sheme često bolja opcija.

4.1. Usporedba ACID i BASE modela

Postoje dva modela kontrole nad transakcijama kako bi se osigurala konzistentnost: ACID, koji se koristi u relacijskim sustavima i BASE kojeg koristi većina NoSQL sustava. Oba modela pružaju način za upravljanje transakcijskim integritetom. Razlika između ova dva modela je sloj u kojem se nalaze, ACID model nalazi se u sloju baze dok je BASE u aplikacijskom sloju, a time i koliki teret leži na programerima baze ili aplikacije.

[11] Schemaless Databases (3.3.). Sadalage, P.J.; Fowler, M. NoSQL Distilled. Addison-Wesley. 2013.

4.1.1. ACID

Česti primjer ACID transakcije je prebacivanja sredstava s jednog bankovnog računa na drugi. Novac se mora oduzeti s jednog računa i dodati na drugi istovremeno u jednoj transakciji. Bitno je da se te dvije opracije obave kao jedna, u potpunosti ili nikako. Ne bi bilo poželjno da se sustav sruši nakon što se novac oduzme s prvog računa, ali prije nego što se prebaci na drugi. Relacijski sustavi značajni su upravo zbog svoje pouzdanosti u upravljanju financijskim transakcijama. Progamerima je lako označiti gdje transakcija počinje i završava. Ako jedna od operacija ne uspije, obje će se vratiti na početak i neće doći do promjene. Softver također osigurava da sustav ne pošalje izvještaj usred transakcije, što znači da se neće dogoditi da vidimo kako je novac nestao i onda se opet pojavio. Ako se izvještaj pokrene dok traje transakcija, on se blokira dok svi dijelovi transakcije ne završe.

Upravljanje transakcijama se odvija na sloju baze. Developeri aplikacija moraju znati samo što učiniti u slučaju da transakcija ne uspije ili kako nastaviti pokušavati dok se uspješno ne izvrši. Nije potrebno brinuti oko poništavanja transakcije jer je to ugrađeno u bazu.[12]

McCreary i Kelly definiraju ACID akronim ovako[13]:

***A**tomicity* (engl. atomarnost) – traksakcija se mora obaviti u potpunosti ili nikako. Sustavi koji osiguravaju atomarnost moraju biti spremni na sve moguće probleme poput softverskih greški, hardverskih problema, kvarova na mreži, zakazivanja diskova ili rušenja cijelog sustava.

***C**onsistency* (engl. konzistentnost) – podaci moraju biti dosljedni. U primjeru s bankovnim računima ukupan se iznos sredstava ne smije mijenjati. Baza ne smije dati izvještaj koji pokazuje da su sredstva povučena s jednog računa, ali nisu dodana na drugi, svi izvještaji moraju se blokirati tijekom atomarnih operacija. Također, veliki broj atomarnih operacija i izvještaja koji se izvode nad istim podacima u bazi ima utjecaj na brzinu sustava.

---

[12] Comparing ACID and BASE—two methods of reliable database transactions (2.5.). McCreary, D; Kelly, A. Making Sense of NoSQL. Manning. 2014.

[13] RDBMS transaction control using ACID (2.5.1.). McCreary, D; Kelly, A. Making Sense of NoSQL. Manning. 2014.

***I**solation* (engl. izoliranost) – operacije su međusobno izolirane, tj. niti jedan dio transakcije nije svjestan ostalih. Operacija koja dodaje sredstva na novi račun ne zna da postoji druga koja prethodno oduzima sredstva s prvog računa.

***D**urability* (engl. trajnost) – Nakon što svi dijelovi transakcije završe, ona je trajna. Ako se bankovni sustav sruši i baza se mora ponovo pokrenuti sa sigurnosne kopije, mora se vratiti i izvještaj o transakciji. Obično se zapisi o transakcijama drže na odvojenom sustavu i provjeravaju nakon što se baza ponovo podigne.

Softver koji upravlja svim navedenim pravilima prilično je kompleksan i jedan je od razloga zbog kojeg RDBMS sustavi mogu biti jako skupi. Iako je podržavanje ACID transakcija skupo, postoje mnoge poznate strategije za njihovo korištenje. Bazirane su uglavnom na zaključavanju resursa, kreiranju dodatnih kopija tih resursa, obavljanje transakcije, i ako sve prođe u redu, ponovno otključavanje resursa. Ako bilo koji dio transakcije zakaže, originalni resurs se mora vratiti u početno stanje. Izazov je napraviti sustave koji podržavaju ACID transakcije, omogućiti aplikaciji lako korištenje transakcija, te održati brzinu odaziva baze.

ACID sustavi fokusiraju se prije svega na konzistentnost i integritet podataka. Privremeno blokiranje izvještaja je razuman kompromis za osiguravanje točnih i pouzdanih informacija. ACID sustavi nazivaju se pesimističnima zbog toga što moraju predvidjeti sve moguće stvari koje mogu poći po zlu i biti spremni na njih.[14]

4.1.2. BASE

Dok ACID sustavi osiguravaju visok integritet podataka, BASE model uzima u obzir druga ograničenja. Postoje situacije i potrebe u kojima nije prihvatljivo da se blokira jedna transakcija dok se druga ne izvrši. Web trgovine bi brzo propale ukoliko bi kupci morali čekati na red i obavljati svoje narudžbe jedan po jedan, tj. da je svim ostalim potencijalnim kupcima blokirana mogućnost kupovanja dok ostali ispred njih ne završe. U tom slučaju ACID sustavi nisu ono što želimo.

[14] RDBMS transaction control using ACID (2.5.1.). McCreary, D; Kelly, A. Making Sense of NoSQL. Manning. 2014.

Web trgovine imaju drugačije prioritete vezane uz transakcije. Problem izvještaja koji su nekonzistentni na nekoliko minuta je manje važan od blokiranja narudžbe, jer bi spriječavanje narudžbe moglo odvratiti kupce. BASE se koristi kao alternativa ACID modelu, a kratica je za[15]:

***B**asically **A**vailable* (engl. uglavnom dostupan) – sustavu se dozvoljava da bude privremeno nekonzistentan kako bi se transakcije mogle izvršavati. Zbog toga se informacije i usluge smatraju uglavnom dostupnima.

***S**oft state* (engl. meko stanje) – neke netočnosti su kratkotrajno dopuštene i podaci se mogu mijenjati dok se koriste kako bi se smanjila potrošnja resursa.

***E**ventually consistent* (engl. u konačnici konzistentan) – nakon što se sve operacije izvrše, sustav će u konačnici biti konzistentan.

Za razliku od ACID modela koji je orijentiran na koznistentnost, sustavi koji koriste BASE model se fokusiraju na dostupnost. Primarni cilj BASE sustava je dopustiti unos novih podataka, čak i ako ti podaci privremeno budu nesinkronizirani. Izvještaji su dopušteni iako svi dijelovi bazi nisu još međusobno sinkronizirani. BASE sustavi nazivaju se optimističnima jer pretpostavljaju da će svi dijelovi sustava u konačnici nadoknaditi zaostatke i postati konzistentnima.

BASE sustavi su obično jednostavniji i brži jer nije potreban kod koji bi se bavio zaključavanjem i otključavanjem resursa. Cilj im je održati tok procesa i baviti se zaostacima i greškama kasnije. To ih čini idealnima za web trgovine gdje je glavni prioritet odabrati artikle i obaviti narudžbu.

Prije NoSQL pokreta, ACID model smatrao se jedinim koji se može koristiti u poslovne svrhe. No NoSQL sustavi su jako decentralizirani i strogosti ACID modela nisu nužna, stoga se koristi fleksibilniji BASE model.

[15] Non-RDBMS transaction control using BASE (2.5.2.). McCreary, D; Kelly, A. Making Sense of NoSQL. Manning. 2014.

Sadalage i Fowler smatraju BASE akronim forsiranim i slabo definiranim[16], ali ukazuju na to da upotreba ACID ili BASE modela nije binarna odluka, odnosno kako se ACID i BASE modeli u nekim situacijama mogu koristiti naizmjenično. Postoje sustavi koji mogu implementirati oba modela te koristiti jedan ili drugi ovisno o potrebama u pojednom trenutku.

[16] The CAP Theorem (5.3.1.). Sadalage, P.J.; Fowler, M. NoSQL Distilled. Addison-Wesley. 2013.

5. Modeli distribucije

Jedno od glavnih svojstava NoSQL baza je mogućnost rada na klasterima. Kako se baza povećava, vertikalno skaliranje postaje sve skuplje, stoga je horizontalno skaliranje na klasterima isplativija opcija. No to unosi druge kompleksnosti. Postoje dva osnovna modela distribucije: replikacija i sharding. Replikacija kopira iste podatke na više čvorova dok sharding razdjeljuje podatke na različite čvorove. Ova dva načina distribucije nisu međusobno isključiva, mogu se koristiti i oba istovremeno.

Sadalage i Fowler ističu kako je najbolji način distribucije - jedan server, odnosno kako ne bi trebalo distribuirati bazu na klastere ako se to ikako može izbjeći.[17] Pokretanje baze na jednom serveru eliminira sve kompleksnosti i probleme koje distribucija unosi, stoga uvijek vrijedi razmotriti tu opciju prije nego što se odluči koristiti distribuirani sustav.

Sharding distribuira podatke po čvorovima u klasteru. Ukoliko količina informacija koju je potrebno pohraniti nadilazi kapacitet sustava, podaci se "razlome" na manje dijelove (engl. shard - krhotina) i rasporede po klasteru. Problem je organizirati podatke tako da podaci koji zahtjevaju zajednički pristup budu na istom čvoru, kako bi se omogućio najbrži pristup. Ovdje do izražaja dolaze agregatno-orijentirane baze. Ako je npr. agregat baziran na fizičkoj lokaciji, podatke možemo staviti na čvorove koji su najbliži toj lokaciji, ili ih rasporediti po nekom drugom kriteriju ovisno o potrebi. Moguće je i ravnomjerno raspodijeliti podatke po svim čvorovima, no to može varirati s vremenom kako se podaci u bazi mijenjaju i ovisi o specifičnim potrebama. Agregati se često grupiraju na čvorove ako imaju određen redosljed čitanja, npr. abecedno. Sharding se može implementirati u kodu aplikacije, no mnoge NoSQL baze imaju auto-sharding u kojem baza automatski preuzima odgovornost raspoređivanja podataka.[18] Sharding je važna tehnika za brzinu rada baze jer može poboljšati i čitanje i zapisivanje podataka. No sam po sebi sharding pruža slabu otpornost na kvarove, ukoliko jedan čvor prestane raditi, podaci na njemu postaju nedostupni. Sposobnost baze da nastavi s

[17] Single Server (4.1.). Sadalage, P.J.; Fowler, M. NoSQL Distilled. Addison-Wesley. 2013.

[18] Sharding (4.2.). Sadalage, P.J.; Fowler, M. NoSQL Distilled. Addison-Wesley. 2013.

radom kada postoje greške u čvorovima ili mreži naziva se tolerancija na odjeljenje (engl. partition tolerance).

Postoje dvije vrste replikacije: master-slave i peer-to-peer. Kod master-slave replikacije jedan čvor je primaran (master) i on je odgovoran za zapisivanje i ažuriranje podataka. Ostali čvorovi su sekundarni (slave) i sinkroniziraju podatke s masterom. Master-slave replikacija se uglavnom koristi kod baze s intenzivnim zahtjevima za čitanje. Lako se horizontalno skalira dodavanjem novih čvorova i osiguravanjem da svi zahtjevi za čitanje idu kroz slave čvorove. To znači da je master čvor usko grlo kod zapisivanja, jer svi zahtjevi moraju proći kroz master koji ih onda mora proslijediti slave čvorovima. No u skladu s time je ovakav način replikacije otporan na greške što se tiče čitanja. Ukoliko dođe do greške u masteru, slave čvorovi i dalje mogu odgovarati na zahtjeve za čitanjem. Također, ako je neki slave čvor replika mastera može postati novi master i tako ubrzati oporavak sustava. Novi master može se odrediti ručnim konfiguriranjem ili klaser može automatski odabrati jedan čvor kao novog mastera. Još jedan nedostatak master-slave replikacije je moguća nekonzistentnost kod čitanja ukoliko se ažurirani podaci nisu još proširili do svih slave čvorova.[19]

U peer-to-peer replikaciji su svi čvorovi jednaki, odnosno svi mogu obraditi zahtjeve za zapisivanjem i čitanjem. Ovaj model replikacije rješava problem uskog grla kod zapisivanja podataka, jer su svi čvorovi ekvivalentni i masteru i slaveu. Također, mogu se jednostavno dodavati novi čvorovi kako bi se unaprijedio učinak sustava. Najveća komplikacija je nekonzistentnost u zapisivanju. Ako se podaci mogu zapisati na više različitih mjesta, postoji mogućnost da jedan zapis bude ažuriran dva ili više puta istovremeno. Moguće je koordinirati replike tako da većina njih potvrdi zapis kako bi osigurali da nema konflikta, no to stvara dodatno opterećenje na mreži.[20]

Sharding i replikacija se mogu kombinirati. Master-slave replikacija u kombinaciji sa shardingom znači da u klasteru postoji više mastera, ali svaki upravlja jednom jedinicom podataka. Column-family baze često koriste sharding u kombinaciji s peer-to-peer

[19] Master-Slave Replication (4.3.). Sadalage, P.J.; Fowler, M. NoSQL Distilled. Addison-Wesley. 2013.

[20] Peer-to-Peer Replication (4.4.). Sadalage, P.J.; Fowler, M. NoSQL Distilled. Addison-Wesley. 2013.

replikacijom. Kako bi se podaci osigurali u slučaju kvarova, uglavnom se koristi replikacijski faktor od 3, što znači da je svaki shard repliciran na barem tri čvora.[21]

5.1. Konzistentnost

Konzistentnost je velik izazov distribuiranih sustava, a problematiku možemo gledati iz više perspektiva.

Konzistentnost ažuriranja odnosi se na izbjegavanje ili rješavanje konflikta zapisa do kojeg dolazi ako isti podatak ažuriraju dvije ili više strana istovremeno. Postoje dva pristupa rješavanja konflikta zapisa: tzv. pesimističan i optimističan. Pesimističan pristup fokusiran je na izbjegavanje konflikta u potpunosti, dok je optimističan orijentiran na njihovo otkrivanje i rješavanje.

Jedna od najkorištenijih pesimističnih metoda je zaključavanje zapisa.[22] Kako bi se podaci mogli ažurirati, potreban je ključ, a sustav osigurava da ključ ne mogu dobiti dva ili više klijenta istovremeno. Čest optimističan pristup je testiranje podataka kojeg klijent želi ažurirati kako bi se vidjelo je li u međuvremenu već ažuriran. Još jedna optimistična metoda je spremiti oba zapisa i označiti da su u konfliktu. Zapisi u konfliktu se onda moraju razriješiti, ručno ili automatski. Pesimističan pristpu se obično preferira jer su konflikti izrazito nepoželjni, no pesimističan pristup može značajno usporiti odaziv baze do te mjere da je neupotrebljiva. Stoga je potrebno je voditi računa o odnosu između sigurnosti (spriječavanje grešaka i konflikta) i brzini odaziva.

Konzistentnost čitanja odnosi se na izbjegavanje ili spriječavanje konflikta između zapisivanja i čitanja podataka. Može se dogoditi situacija da jedan klijent čita određene podatke dok ih drugi istovremeno ažurira. U tom slučaju podaci neće biti međusobno konzistentni dok ažuriranje ne završi. Relacijske baze izbjegavaju ovakvu situaciju pomoću transakcija. Kada je zapis obavljen transakcijom, ona osigurava da će klijent pročitati podatke prije transakcije ili nakon nje, pa će podaci biti usklađeni u oba slučaja. Graf baze podržavaju ACID transakcije, stoga izbjegavaju ovakvu situaciju. Agregatno-orijentirane baze podržavaju

[21] Combining Sharding and Replication (4.5.). Sadalage, P.J.; Fowler, M. NoSQL Distilled. Addison-Wesley. 2013.

[22] Update Consistency (5.1.). Sadalage, P.J.; Fowler, M. NoSQL Distilled. Addison-Wesley. 2013.

atomarna ažuriranja, ali samo unutar agregata.[23] To osigurava konzistentnost podataka unutar istog agregata, ali moguća je nekonzistentnost između agregata.

Replikacija unosi još jedan problem. Potrebno je osigurati da su replike međusobno konzistentne, odnosno da isti podatak ima istu vrijednost ako se čita s različitih replika. Kod replikacije će se ažurirani zapis u konačnici proširiti do svih replika, pa se ovakva situacija naziva konačna konzistentnost (engl. eventual consistency).

Bitno je također napomenuti kako je konzistentnost promjenjivo svojstvo kojem je moguće podesiti potrebnu razinu za pojedine zahtjeve, što omogućava da osiguramo visoku razinu konzistentnosti kad je ona najpotrebnija.

Konzistentnost je očito uvijek poželjna, no u nekim situacijama potrebno je donekle žrtvovati konzistentnost kako bi osigurali druge prednosti. Eric Brewer 2000. je godine osmislio tzv. CAP teorem. On tvrdi da od tri svojstva u distribuiranom sustavu, konzistentnost (engl. consistency), dostupnost (engl. availability) i tolerancija na odjeljenje (engl. partition tolerance), moguće je u potpunosti osigurati samo dva.[24] Dostupnost u kontekstu CAP teorema odnosi se na mogućnost komunikacije s određenim čvorom, odnosno ako čvor može zapisivati i čitati podatke - smatra se dostupnim. Tolerancija na odjeljenje odnosi se na sposobnost sustava da preživi prekide u vezi koji razdvajaju klaster na više dijelova (particija) koji međusobno ne mogu komunicirati.

Kelly i McCreary definiraju CAP teorem ovako:[25]

Consistency (engl. konzistentnost) – osiguravanje ažurirane, čitljive verzije podataka dostupne svim klijentima. Ova vrsta konzistentnosti nije ona o kojoj govori ACID model. Ovdje se konzistentnošću smatraju rezultati koji klijenti dobivaju pri čitanju istih podataka sa repliciranih particija.

---

[23] Read Consistency (5.2.). Sadalage, P.J.; Fowler, M. NoSQL Distilled. Addison-Wesley. 2013.

[24] The CAP Theorem (5.3.1.). Sadalage, P.J.; Fowler, M. NoSQL Distilled. Addison-Wesley. 2013.

[25] Understanding trade-offs with Brewer's CAP theorem (2.7.). McCreary, D; Kelly, A. Making Sense of NoSQL. Manning. 2014.

Availability (engl. dostupnost) – distribuirana baza uvijek bi trebala dopuštati klijentima ažuriranje podataka bez odgađanja. Greške u internoj komunikaciji između repliciranih podataka ne bi trebale spriječiti ažuriranje.

Partition tolerance (engl. tolerancija na odjeljenje) – sposobnost sustava da odgovara na zahtjeve klijenata iako postoje greške u komunikaciji između particija baze.

Sustav s jednim serverom primjer je CA sustava. Osigurava konzistentnost i dostupnost, ali s obzirom na to da se radi o samo jednom serveru, sustav nije sposoban izdržati bilo kakve greške ili prekide u radu. Ako je server u kvaru – baza ne radi. Relacijske baze uglavnom rade po ovom principu.

Komunikacijski prekidi neizbježni su u distribuiranim sustavima. Iako CAP teorem tvrdi da se mogu osigurati samo dva od tri svojstva, u praksi je potrebno osigurati toleranciju na odjeljenje i pritom odlučiti treba li žrtvovati dostupnost ili konzistentnost. No izbor između dostupnost i konzistentnosti nije binarna odluka, već je moguće podesiti razine dostupnosti i konzistentnoti, ovisno o potrebama i situaciji.[26]

Kako bi osigurali visoku razinu konzistentnosti kod peer-to-peer replikacije, nije potrebno uključiti sve čvorove u zahtjev za čitanjem ili pisanjem, ali je potrebna većina. Ako imamo tri čvora s repliciranim podacima, dva su dovoljna da potvrde zapis i riješe konflikt. Kvorum zapisivanja glasi W>N/2, gdje je W broj čvorova koji zapisuje podatak, a mora biti veći od N koji predstavlja broj čvorova koji sudjeluju u replikaciji (faktor replikacije). Kvorum čitanja je R+W>N, gdje je R broj čvorova koje trebamo kontaktirati za čitanje, W je broj čvorova koji potvrđuju zapis, a N je faktor replikacije. Ako imamo 3 čvora, dovoljan je odgovor dvaju njih kako bi potvrdili ažuriranost podataka.[27]

[26] The CAP Theorem (5.3.1.). Sadalage, P.J.; Fowler, M. NoSQL Distilled. Addison-Wesley. 2013.

[27] Quorums (5.5.). Sadalage, P.J.; Fowler, M. NoSQL Distilled. Addison-Wesley. 2013.

6. Pregled NoSQL baza

6.1. Riak

Key-value baza jednostavna je tablica koja pohranjuje vrijednosti tako da im dodijeli jedinstveni ključ. S aplikacijske strane, key-value baze su najjednostavnija vrsta NoSQL baza. Klijent može dohvatiti vrijednost pomoću ključa, dodijeliti ključ novoj vrijednosti i izbrisati ključ iz baze, a time i podatak na koji taj ključ upućuje.

Riak je open-source baza napisana u programskom jeziku Erlang. U osnovi je key-value baza podataka, ali sadrži i neke funkcije dokumentnih baza, zbog čega je Rick Cattell (2011) naziva naprednom key-value bazom.[28]

Riak segmentira ključeve tako da ih pohrani u tvz. "bucket" (engl. kante) - jednostavno prostor u koji se sprema naziv ključa. Bucketima se može dodijeliti određena domena (engl. domain buckets) za pohranu objekata koje se želi na neki način povezati ili grupirati.

Riak objekti mogu se pohraniti u JSON formatu i stoga imati više polja, poput dokumenata, a način na koji se objekti mogu grupirati u buckete sličan je grupiranju dokumenata u kolekcije ili zbirke kod dokumentnih baza. Međutim, moguće je indeksirati isključivo primarne ključeve, dok mehanizmi upita koji postoje kod dokumentnih baza nisu podržani. Riak ima jedinstveno svojstvo pohranjivanja poveznica (engl. links) između objekata, kako bih se lakše kategoriziralo.

Primjer objekta u Riaku:

```
{
"bucket": "knjige",
"key": "123abc",
"object":{ "naziv": "Zov divljine",
        "autor": "Jack London",
        "godina": "1903." }
"links": ["američka književnost", "naturalizam"]
}
```

[28] Riak (2.2.). Cattell, R. Scalable SQL and NoSQL Data Stores. 2011.

Riak podržava sharding i replikaciju objekata, te dopušta privremenu nekonzistentnost kod replika. Razina konzistentnosti može se podesiti određivanjem broja replika koji moraju odgovoriti da bi se zapisivanje ili čitanje smatralo uspješnim.[29] Moguće je i posebno podesiti razinu konzistentnosti za svaki pojedini zapis odnosno čitanje, što daje više fleksibilnosti, ovisno o pojedinim zahtjevima aplikacije.

Upiti se mogu izvršavati isključivo nad ključem, odnosno nije moguće koristiti bazu preko atributa koju neka vrijednost sadrži. Ključ se može generirati pomoću algoritma, pružiti korisniku na temelju ID-a ili email adrese, ili izvesti iz vremenske oznake (engl. timestamp) ili drugih podataka izvan baze. Način na koji se oblikuje ključ utječe na mogućnost i način izvršavanja upita.

Kao i kod ostalih key-value baza, struktura podataka koji se pohranjuju u bazu nije definirana. Vrijednost u Riaku može biti u bilo kojem obliku, tekstu, URL-u, XML-u, JSON-u i sl. Moguće je radi preglednosti i snalaženja označiti vrstu podataka sa "Content-Type".

Riak, kao i većina key-value baza, koristi sharding tehniku za potrebe skaliranja. Vrijednost ključa utječe na čvor na koji se taj ključ pohranjuje kao i kriterij po kojem se odvija sharding. Ključevi se mogu dijeliti abecedno na temelju prvog znaka, na primjer ključevi T55R74 i T94F72 biti će pohranjeni na istom čvoru, dok će ključ V27K14 biti pohranjen na nekom drugom. Na ovaj način može se lako poboljšati izvedba sustava dodavanjem novih čvorova u klaster. Međutim, ukoliko čvor koji pohranjuje "T" ključeve prestane raditi, podaci pohranjeni na tom čvoru nisu više dostupni, niti se mogu zapisati novi podaci kojima je dodijeljen ključ se istim početnim znakom.

Upotreba

Riak je iznimno koristan za jednostavne aplikacije s visokom stopom zapisivanja i čitanja. Idealan je za pohranu sesijskih podataka, što i jest originalno bila svrha baze.[30] S ozbirom da Riak ne postavlja ograničenja u vezi vrste podataka koji se pohranjuje u bazu, informacije se mogu enkodirati na razne načine i mijenjati bez promjene sheme. Sve vezano uz jednu sesiju

[29] Riak (2.2.). Cattell, R. Scalable SQL and NoSQL Data Stores. 2011.

[30] Use Cases. Riak Docs. URL: http://docs.basho.com/riak/latest/dev/data-modeling/ (10.4.2015).

može se pohraniti u jedan objekt i vrlo brzo unijeti u bazu PUT naredbom, odnosno dohvatiti GET naredbom. Podržan je i automatski istek ključeva nakon određenog vremena, kao i MapReduce operacije, što omogućuje i nešto kompleksnije radnje poput obrade i analize senzornih podataka. Riak se često koristi i za pohranu korisničkih računa, tako što se na primjer korisnički objekti pohrane u jedan bucket, gdje korisničko ime može poslužiti kao ključ. U korisnički objekt se također mogu pohraniti i korisničke postavke i preference.

6.2. MongoDB

Dokumente baze podržavaju nešto kompleksnije oblike podataka od key-value baza. Ovaj tip baza se može promatrati kao key-value baza u kojem su vrijednosti transparentne, odnosno mogu se pregledavati i pretraživati. Struktura pojedinih dokumenata u bazi može se razlikovati, ali ti dokumenti svejedno mogu pripadati istoj kolekciji, za razliku od relacijskih baza u kojima svaki red u tablici mora slijediti istu shemu. Svaki stupac u tablici relacijske baze mora biti definiran. Novi atributi u dokumentnoj bazi mogu se stvarati bez potrebe za prethodnim definiranjem ili mijenjanjem postojećih dokumenata.

MongoDB je open-source baza napisana u C++-u. Svaka instanca MongoDB-a tehnički ima više baza, a svaka baza sadrži određeni broj kolekcija. Ako usporedimo s relacijskim sustavima, jedna MongoDB baza (unutar instance) odgovara shemi u relacijskom sustavu, dok su MongoDB kolekcije ekvivalentne relacijskim tablicama.[31]

MongoDB pohranjuje podatke u binarnom formatu sličnom JSON-u, nazvan BSON, koji podržava boolean, integer, float, date, string, i binarne tipove podataka.[32]
Upiti se izražavaju u JSON formatu. Kao i kod većine dokumentnih baze, MongoDB omogućuje izvršavanje upita nad podacima unutar dokumenta (odnosno dijelom dokumenta) ukoliko znamo njegov ključ, bez potrebe za povratkom cijelog dokumenta iz baze.
Ovako izgleda dohvat svih dokumenata iz jedne zbirke u MongoDB-u u usporedbi s vraćanjem svih redova iz relacijske tablice:

[31] Features (9.2.). Sadalage, P.J.; Fowler, M. NoSQL Distilled. Addison-Wesley. 2013.

[32] MongoDB (3.3.). Cattell, R. Scalable SQL and NoSQL Data Stores. 2011.

RDBMS: SELECT * FROM knjige
MongoDB: db.knjige.find()

Sva djela jednog autora, preko njegovog ID-a:
RDBMS: SELECT * FROM knjige WHERE autorID = "123abc"
MongoDB: db.knjige.find({"autorID": "123abc"})

Dohvat ID-a knjige i datuma izdavanja:
RDBMS: SELECT knjigaID, knjigaDatum FROM knjige WHERE autorID = "123abc"
MongoDB: db.knjige.find({autorID: "123abc"}, {knjigaID:1, knjigaDatum:1})

Konzistentnost u MongoDB-u ostvaruje se master-slave replikacijom. Replikacija se odvija na razini sharda, a baza podržava automatski sharding preko ključa kojeg definira korisnik. Ne postoji globalna konzistentnost kao kod relacijskih baza, ali se može ostvariti lokalna konzistentnost pojedinog dokumenta. Vrijedi princip kvoruma, odnosno setovi replika konfiguriraju se tako da se zapisi repliciraju do određenog broja slave čvorova kako bi se zapis smatrao uspješnim ili određeni broj čvorova mora odgovoriti na zahtjev u slučaju čitanja. Veća razina konzistentnosti znači i sporije zapise, jer treba više vremena da se zapisi propagiraju do svih čvorova. Brzina čitanja može se povećati tako da se omogući čitanje sa slave čvorova, što se može omogućiti na razini cijele baze, određene kolekcije ili za svaku operaciju pojedinačno.[33] Replikacijom se osigurava i dostupnost. Klijent može pristupiti podacima na drugim čvorovima iako glavni čvor ne reagira. Najmanje dva čvora sudjeluju u asinkronoj replikaciji podataka. Čvorovi u setu replika međusobno odabiru primarni čvor (master) na temelju kriterija poput blizine ostalim čvorovima, više memorije, i sl. Prioritet čvorovima može se odrediti i ručno, brojevima 0 - 1000. Ako master čvor prestane funckionirati, ostali čvorovi međusobno odabiru novi čvor koji postaje master. Kad se prijašnji master čvor vrati u funkciju, pridruži se kao slave.

Transakcije slične onima u relacijskim bazama ne postoje, ali MongoDB podržava atomarne operacije. Ažuriranje pojedine vrijednosti "update" naredbom može se ostvariti

[33] Consistency (9.2.1.). Sadalage, P.J.; Fowler, M. NoSQL Distilled. Addison-Wesley. 2013.

modifikatorima poput $set - postavlja vrijednost, $inc - povećava vrijednost, $push - dodaje vrijednost u polje, $pull - izbacuje vrijednost iz polja, i sl.[34]

MongoDB se vrlo lako skalira dodavanjem novih čvorova. Skaliranje za potrebe velikog opterećenja kod čitanja postiže se jednostavnim dodavanjem više slave čvorova. Novi čvor se pri dodavanju sinkronizira s ostalima, i automatski počinje posluživati zahtjeve za čitanjem. Sve je to moguće napraviti bez prekida u radu aplikacije ili potrebe za ponovnim pokretanjem čvorova. Kod skaliranja za potrebe zapisivanja, koristi se sharding. Sve kolekcije moraju imati index koji počinje sa shard ključem, jer je on osnova po kojoj se sharding odvija.[35] Sharding se može usporediti s particijama u relacijskom sustavu gdje se podaci podijele po vrijednosti u određenom stupcu. No particije se nalaze na jednom serveru, stoga aplikacija ne mora izvršavati upit nad određenom particijom nego samo nad glavnom tablicom, dok baza pronalazi particiju i vraća podatke. Kod shardinga se podaci također podijele po određenoj vrijednosti, ali se onda prebace na različite čvorove. Podaci se dinamički razmještaju po čvorovima kako bi shardovi bili uravnoteženi. Aplikacija pri odvijanju svih promjena može raditi bez prekida, iako klaster možda neće raditi optimalno dok se prebacuju velike količine podataka.

Upotreba

S obzirom na to da nema definiranu shemu i podržava JSON format MongoDB je odličan izbor za CMS (engl. Content Management System), aplikacije za objavu web stranica, upravljanje komentarima korisnika, profile i web dokumente. Također su mogu pohraniti analitički podaci, s obzirom na to da se dijelovi dokumenta mogu ažurirati zasebno, lako je pohraniti broj posjeta stranici i novi metrički podaci se mogu dodati bez promjene sheme. Također može biti valjan izbor i za web trgovinu radi fleksibilne sheme za proizvode i narudžbe.

[34] MongoDB (3.3.). Cattell, R. Scalable SQL and NoSQL Data Stores. 2011.

[35] Shard Keys. MongoDB Manual. URL: http://docs.mongodb.org/manual/core/sharding-shard-key/ (10.4.2015).

6.3. Cassandra

Column-family baze su NoSQL baze kojima se podaci spremaju po principu ključ-vrijednost, s time da su vrijednosti koje odgovaraju jednom ključu grupirane u stupce. Skupine tj. obitelji stupaca ili column-families su skupine srodnih podataka kojima se najčešće pristupa zajedno. Stupci moraju biti prethodno definirani, ali se novi atributi mogu dodati u bilo kojem trenutku.

Cassandra je napisana u Javi, a često se opisuje kao spoj Amazonove baze Dynamo i Googleovog BigTable-a.[36]

Osnovna jedinica za pohranu podataka u Cassandri je stupac koji se sastoji od para imena i vrijednosti, gdje ime ima funkciju ključa. Stupci se uvijek pohranjuju s vremenskim žigom (engl. timestamp) koji služi rješavanje konflikata kod zapisivanja i sl.

Red čini skupina stupaca koji su povezani ključem. Skupina sličnih redova čini column-family ili obitelj stupaca.[37]

```
//obitelj stupaca
{
//red
        "pero-peric": {
                ime: "Pero",
                prezime:"Perić",
                zadnjiPosjet:"2014/04/04"
        }
//red
        "ivo-ivic": {
                ime:"Ivo",
                prezime:"Ivić",
                lokacija:"Zagreb"
        }
}
```

[36] Cassandra (4.3.). Cattell, R. Scalable SQL and NoSQL Data Stores. 2011.

[37] Features (10.2.). Sadalage, P.J.; Fowler, M. NoSQL Distilled. Addison-Wesley. 2013.

Red se proteže kroz više stupaca, a identificira se ključem. Različiti redovi ne moraju imati iste stupce, koji se mogu dodati bilo kojem redu u bilo kojem trenutku a da pritom budu vezani isključivo za taj red a ne i za ostale. Stupci također mogu u sebi sadržavati i druge stupce (engl. super column), a takvi nad-stupci mogu stvoriti novu obitelj stupaca (engl. super column family). Ovako složene razine podataka mogu pogodovati velikoj količini povezanih podataka, ali to znači da se svi vraćaju zajedno iako nisu nužno potrebni, što nije optimalno.

Keyspace je prostor koji Cassandra pripisuje obitelji stupaca koji su vezani za određenu aplikaciju. Cassandra pohranjuje zapise u svoju unutrašnju memoriju nazvanu *memtable*. Zapisi se smatraju uspješnima tek kada su pohranjeni u memtable. Zapisi u Cassandri odvijaju se atomarno na razini reda, što znači da se dodavanje novih ili ažuriranje postojećih stupaca vezanih uz jedan ključ tretira kao jedan zapis koji će biti uspješan ili neuspješan.

Razina konzistentnosti može se jednostavno zadati:

```
quorum = new ConfigurableConsistencyLevel();
quorum.setDefaultReadConsistencyLevel(HConsistencyLevel.QUORUM);
quorum.setDefaultWriteConsistencyLevel(HConsistencyLevel.QUORUM);
```
[38]

Postavljanje kvoruma kao razine konzistentnosti osigurava odgovor većine čvorova te vraćanje podataka s najnovijim vremenskim žigom kod čitanja. Postavkom "ALL" može se osigurati maksimalna razina konzistentnosti, ali se time žrtvuje tolerancija na odijeljenje jer svi čvorovi moraju odgovoriti na zahtjeve za čitanjem i zapisivanjem. Čak i ako samo jedan čvor zakaže, operacija se blokira i smatra neuspješnom.

Kada jedan čvor ne radi, podaci koji su trebali biti spremljeni na taj čvor proslijeđuju se drugim čvorovima. Kad se čvor ponovo proradi, podaci se automatski prebacuju natrag na njega. Ova tehnika naziva se *hinted handoff* i ubrzava vraćanje pokvarenih čvorova u funkciju.[39]

Cassadra koristi peer-to-peer replikaciju koja osigurava visoku dostupnost. Dostupnost klastera može se dodatno povisiti smanjenjem razine konzistentnosti zahtjeva.

---

[38] Consistency (10.2.1.). Sadalage, P.J.; Fowler, M. NoSQL Distilled. Addison-Wesley. 2013.

[39] Consistency (10.2.1.). Sadalage, P.J.; Fowler, M. NoSQL Distilled. Addison-Wesley. 2013.

Prije izvršenja upita nad bazom, potrebno je navesti keyspace koji koristimo, npr. "use autori;"

Cassandra koristi vlastiti jezik za upite koji je sličan SQL-u, a naziva se Cassandra Query Language - CQL. Naredbe su gotovo identične kao u SQL-u, ali bez nekih mogućnosti poput joinova i podupita.

```
CREATE COLUMNFAMILY Knjige (
        KEY varchar PRIMARY KEY,
        naslov varchar,
        autor varchar);


INSERT INTO Knjige (KEY,naslov,autor)
        VALUES ('zovd',
        'Zov divljine',
        'Jack London' );


SELECT naslov,autor FROM Knjige
```

Skaliranje postojećeg klastera u Cassandri postiže se jednostavnim dodavanjem više čvorova. S obzirom na to da su u peer-to-peer klasteru svi čvorovi jednaki, klaster je sposoban obraditi više zahtjeva za čitanjem i pisanjem što je više čvorova u njemu. Ovaj tip horizontalnog skaliranja omogućuje neprekidan rad jer ostatak klastera nastavlja posluživati zahtjeve klijenata dok se istovremeno uključuju novi čvorovi.[40]

Upotreba

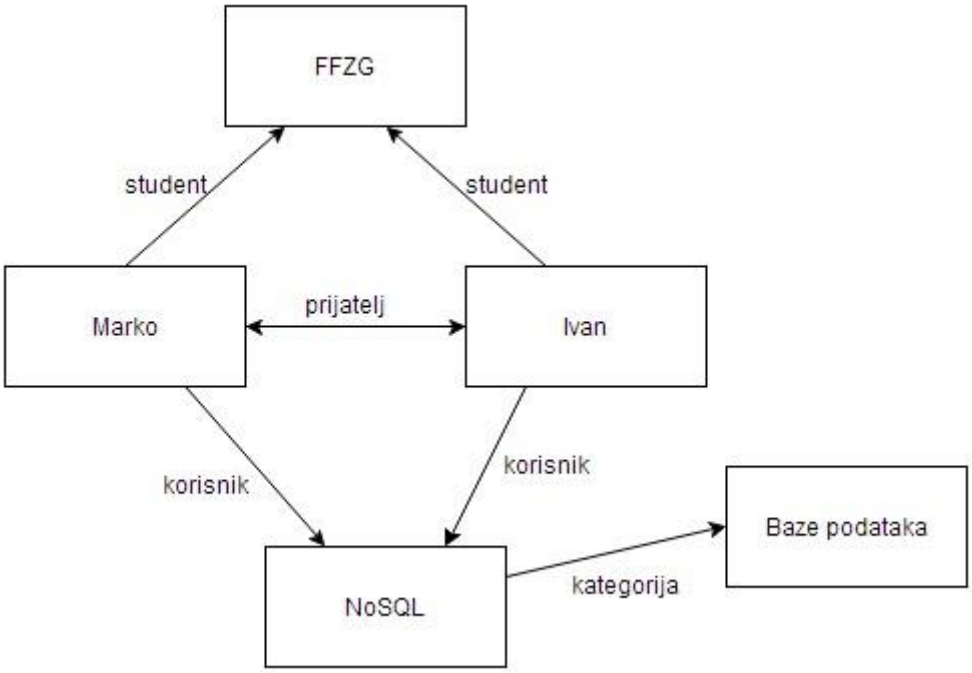
Casssandra je prigodna kao baza za razne oblike elektroničkih poruka, od emaila, chata, komentara i sl. Takva baza zahtjeva visoku razinu skalabilnosti kako bi se mogla nositi s

[40] Scaling (10.2.5.). Sadalage, P.J.; Fowler, M. NoSQL Distilled. Addison-Wesley. 2013.

podacima sa raznih uređaja i platformi.[41] Cassandra je uspjeh na tom području dokazala time što je koristi i Facebook.[42]

6.4. Neo4j

Graf baze pohranjuju entitete i njihove međusobne poveznice. Entiteti su čvorovi koji imaju svojstva, a mogu se promatrati kao instanca objekta u aplikaciji. Poveznice, odnosno veze među entitetima nazivaju se još rubovi ili bridovi (engl. edges) i također imaju vlastita svojstva. Poveznice imaju određen smjer koji je bitan za pronalazak različitih obrazaca i interpretaciju podataka.



[41] Apache Cassandra For Messaging. Planet Cassandra. http://planetcassandra.org/blog/functional_use_cases/messaging/ (10.4.2015).

[42] Cassandra (4.3.). Cattell, R. Scalable SQL and NoSQL Data Stores. 2011.

Svojstva čvorova i poveznica omogućuju lakšu organizaciju. Poveznice mogu imati više svojstava. Bitan je i smjer relacije, "prijatelj" je dvosmjerna relacija, dok "korisnik" nije. Upiti se kreiraju na temelju grafa, npr. možemo zatražiti bazu da nam prikaže sve studente FFZG-a koji koriste NoSQL. Izvršavanje upita nad grafom naziva se i putovanje po grafu (engl. graph traversal).[43] Po grafu se može putovati na bilo koji način, kao i dodavati nove čvorove i poveznice, bez promjene modela podataka. Poveznice se ne izračunavaju prilikom svakog upita nego su stalne. Svojstva čvorova i poveznica mogu se indeksirati.

Neo4j je open-source baza napisana u Javi. Stvaranje grafa u bazi Neo4J vrlo je jednostavno, dovoljno je kreirati dva čvora i povezati ih:

```
Node marko = graphDb.createNode();

marko.setProperty("ime", "Marko");

Node ivan = graphDb.createNode();

ivan.setProperty("ime", "Ivan");
```

Kada postoje barem dva čvora, može se napraviti poveznica:

```
marko.createRelationshipTo(ivan, PRIJATELJ);

ivan.createRelationshipTo(marko, PRIJATELJ);
```

U ovom sluačaju je potrebno stvoriti dvosmjernu poveznicu. Smjer je ključan za razumjevanje odnosa između dva čvora, npr. korisniku se može sviđati neki proizvod, ali korisnik se ne može sviđati proizvodu. Poveznice osim smjera, polazišnog i odredišnog čvora mogu imati i razna svojstva. Mogu sadržavati vrstu odnosa dviju osoba kao u gornjem primjeru, udaljenost između čvorova (kod baze s gradovima recimo), druge aspekte koje čvorovi dijele i sl. Upiti se mogu izvršavati i na temelju svojstava poveznica.

[43] What is a Graph Database? (11.1.). Sadalage, P.J.; Fowler, M. NoSQL Distilled. Addison-Wesley. 2013.

Neo4j u potpunosti podržava ACID svojstva. Za rad na klasteru koristi se master-slave replikacija. Zapis na masteru se nakon nekog vremena sinkronizira sa slave čvorovima, koji su uvijek dostupni za čitanje. Dopušteno je i zapisivanje na slave čvor koji se onda sinkronizira s masterom, čime se osigurava visoka dostupnost, ali ostali slave čvorovi neće biti odmah sinkronizirani nego moraju čekati da do njih dođu podaci s mastera.[44]

Graf baze osiguravaju konzistentnost kroz transakcije, ne dopuštaju poveznice koje "vise" odnosno imaju samo jedan čvor. Početni i završni čvor uvijek moraju postojati, te se čvor može izbrisati tek kada su izbrisane sve njegove poveznice.

S obzirom na to da Neo4j podržava ACID svojstva, promjene u čvorovima ili poveznicama, te dodavanje novih čvorova i poveznica uvijek započinje transakcijom. Način izvođenja transakcije nešto se razlikuje od relacijskih baza. Transakcija se mora označiti kao uspješnom naredbom "success", u suprotnom će se baza vratiti u prijašnje stanje nakon završetka. Također, ako izostavimo završetak transakcije (naredba "finish"), ona neće biti izvršena.[45]

```
Transaction transaction = database.beginTx();
try {
   Node node = database.createNode();
   node.setProperty("knjiga", "Hobit");
   node.setProperty("objavljeno", "1937");
   transaction.success();
} finally {
   transaction.finish();
}
```

[44] Consistency (11.2.1.). Sadalage, P.J.; Fowler, M. NoSQL Distilled. Addison-Wesley. 2013.

[45] Transactions (11.2.2.). Sadalage, P.J.; Fowler, M. NoSQL Distilled. Addison-Wesley. 2013.

Neo4j koristi Cypher kao jezik za upite. Za izvršavanje upita (tj. putovanje po grafu) potreban je početni čvor koji se može dobiti preko ID-a ili indeksa.

```
START FFZG = node:nodeIndex(ime = "FFZG")
MATCH (FFZG)--(connected_node)
RETURN connected_node
```

Možemo izvršiti upit za smjer poveznice na ovaj način:

```
MATCH (FFZG)<--(connected_node)
```

Za ulazne poveznice ili:

```
MATCH (FFZG)-->(connected_node)
```

Za izlazne.

Iako Neo4j podržava sharding, kod graf baza to nije tako jednostavno izvesti zbog ograničenja potrebnih da se zadovolje ACID svojstva. Moguće je umjesto shardinga jednostavno dodati više memorije ukoliko set podataka nije prevelik za razumnu količinu RAM-a. Moguće je lako skalirati za potrebe čitanja dodavanjem više slave čvorova koji su konfigurirani isključivo za čitanje (engl. read-only), dok svi zapisi idu preko mastera.

Ukoliko se ipak koristi sharding, potrebno je obratiti pozornost na raspored čvorova po instancama baze, pogotovo ako su na različitim geografskim lokacijama. Nije dobro preopteretiti jednu lokaciju/instancu čvorovima, dok ih je na ostalima nerazmjerno malo. Isto tako je problem ako su raspoređeni tako da ima puno poveznica koje prelazi iz jedne instance u drugu, jer je potrebno puno vremena za putovanje po takvom grafu. Idealno bi bilo da su čvorovi podjednako raspoređeni i da su pritom minimalne poveznice koje prelaze iz jedne instance u drugu. Međutim to je teško ostvariti, velikim djelom i zato jer se graf može često mijenjati pa idealna situacija ne može dugo opstati.

Upotreba

Neo4j izvrstan je odabir za društvene mreže koje već same po sebi imaju strukturu grafa, stoga ih nije potrebno pretvoriti u tablice kao što bi bio slučaj s ostalim bazama. Može se koristiti kod bilo kakvih podataka koji imaju strukturu grafa, jer je u tom slučaju shema u biti jednaka podacima.[46] To mogu biti lokacijski servisi, koji nude sadržaje u blizini temeljene na sličnosti s prethodnim odabirima, ili servis za preporuku srodnih proizvoda u web trgovinama i sl.

[46] Why Neo4j. Social Network. Use Case. Neo4j. URL: http://neo4j.com/use-cases/social/ (20.04.2015)

7. Zaključak

NoSQL pokret često nailazi na nerazumjevanje i pogrešno se smatra lošom zamjenom za relacijski model. No poanta NoSQL-a je upravo ispuniti one zadatke koje relacijski model ne može baš najbolje riješiti. Svoj proboj na tržištu NoSQL baze ne duguju samo sposobnosti rješavanja određenih problema, nego i podršci velikih tvrtki kao što su Google, Amazon, Facebook i Twitter, koji su razvili i koriste NoSQL proizvode. NoSQL je svoje mjesto očito našao na webu, no to ne znači da je dobra ideja pri izradi web aplikacije automatski uzeti NoSQL bazu. Potrebno je pomno isplanirati sustav, razmisliti o potrebama i vidjeti koje je najbolje moguće rješenje.

NoSQL sustavi sada se razlikuju na razne načine, no pritisak tržišta bi u budućnosti mogao primorati razvoj standardiziranog modela, barem unutar svake od pojedinih vrsta NoSQL baza. Ovakva standardizacija bi uvelike olakšala razvoj novih i unaprijeđenje postojećih sustava, lakši prelazak s jednog sustava na drugi, a i lakše osposobljavanje za programiranje i korištenje sustava.

Iako su zasada svaka od 4 vrste NoSQL baza prilično dobro zastupljene u određenim područjima, preostaje za vidjeti hoće li svaka od njih preživjeti ili će jedna ili više njih izaći iz upotrebe u budućnosti.

8. Literatura

Strauch, C. NoSQL Databases. URL: http://www.christof-strauch.de/nosqldbs.pdf (06.12.2013.)

Kelly, A.; McCreary, D. Making Sense of NoSQL. Manning. 2014.

Tiwari, S. Professional NoSQL. John Wiley & Sons. 2011.

Vaish, G. Getting Started With NoSQL. Packt. 2013.

Sadalge, P.J.; Fowler, M. NoSQL Distilled. Addison-Wesley. 2013.

Cattell, R. Scalable SQL and NoSQL Data Stores. 2011. URL: http://cattell.net/datastores/Datastores.pdf (20.05.2014.)

RiakDocs. URL: http://docs.basho.com/riak/latest/ (12.10.2013.)

MongoDB 3.0 Manual. URL: http://docs.mongodb.org/manual/ (12.10.2013.)

Documentation. Apache Cassandra 2.0. URL: http://docs.datastax.com/en/cassandra/2.0/cassandra/gettingStartedCassandraIntro.html (13.10.2013.)

The Neo4j Manual v2.2.1. URL: http://neo4j.com/docs/2.2.1/ (14.10.2013.)